Overview

在本次的實驗報告中，利用PyTorch實作了SCCNet的架構，透過論文中可以調整的參數，在三個訓練分類中，記錄了超過20萬組的訓練次數，最終SD拿到最高的準確度為65.23%、LOSO為63.54%、LOSO+FT為80.90%。

本實驗報告首先會簡單介紹 Trainer 與Tester實作了哪些功能，接著講解SCCNet的架構，和簡單講解實作方法。接著是在訓練階段根據訓練給的參數做比較，如何調整參數才能達到更佳的準確度。

在本實驗報告的最後一小節則是討論EEG資料集的訓練難度和透過這次實驗能如何去提升它的準確度。

Implementation Details

**Details of training and test code**

在training.py中設計了一個Trainer的Class，在建構子帶入的參數有。

* numClasses：最後輸出的類別數量
* timeSample：每一個時間段(single trials)的特徵值
* Nu：第一層 Conv 的輸出通道數(Out\_Channel)
* C：EEG資料的 C(22) 個檢測點
* Nc：第二層 Conv 的輸出通道數
* Nt：第一層 Conv 的feature map的W大小
* dropout\_rate：設定SCCNet中Dropout的大小
* weight\_decay：設定L2 regularization 大小
* learning\_rate：設定model的學習率
* optimizer\_name：設定optimizer的類型

對於這次實驗報告可以調配的參數有 Nu、Nt、learning\_rate、optimizer等，之外也可以對輸入的訓練集大小做mini-batch的訓練，因此batch\_size也是一個可以調整的參數之一。

此外Trainer的class設計了加載訓練與測試資料集的函數，在加載資料時會需要設定每次在訓練時取資料的batch大小，而在訓練資料集上匯兌資料做亂數打亂的動作，而測試資料集上則不會做打亂的動作。

在訓練時會同時顯示測試資料上的準確度(accuracy)和訓練及本身的單次訓練的loss。

本實驗所使用的loss function為nn.CrossEntropyLoss()，因為此實驗是一個分類問題，且在SCCNet中的最後一層使用的是softmax，將結果轉為機率分布，因此Loss function上採用CrossEntropy的方式。

Trainer同時提供了Train和Evaluate的功能，但是對於Tester只能使用Evaluate，Trainer在train函數的設計上可以將資料儲存到資料庫，做統一的儲存(功能寫至utils.py)，後續要將資料提出或者過濾上利用SQL語法都能很快的抓資料出來(在後續取資料環節都是使用)。

在模型權重的儲存上則是會根據可調配的參數，將會儲存在

{train\_method}/{model\_name}/Nu={Nu}-Nt={Nt}-LR={lr}-Opt={opti\_name}-BZ={batch\_size}/epoch={epoch}.pt

其中train\_method是指本次作業中提到的SD/LOSO/LOSO+FT，model\_name則是額外制定的變數名稱，用來辨識不同的model。

在Tester class的撰寫上都跟Trainer差不多，只是擷取部分功能，能直接抓取特定位置的權重紀錄檔。

**Details of the SCCNet**

SCCNet的架構分成四層。

1. Conv2D
2. Conv2D
3. Pooling
4. Softmax

在第一層的Conv2D，使用個大小為()的feature map，這一層的Conv2D做完之後還會做Batch Normalization和Zero-padding。

在第二層的Conv2D，使用個大小為()的feature map，這一層的Conv2D做完之後還會做Batch Normalization和Zero-padding，這邊的Zero-padding會特別拉到一樣大小的size，但是因為kernel map是偶數，在padding時必定會多一格出來。

在第二層結束後會做SquareLayer去放大特徵值，接著做dropout防止training時出現overfitting的問題。

在第三層的pooling則是使用AvgPool，pooling完畢之後會再做一個Log (Activation) Layer。

做完Log Layer之後會將資料攤平，然後進入fully connect層。

Fully connect層出來之後會得到 4 個classes的輸出，接著進到最後一層softmax，實驗上使用F.log\_softmax()，提升在training時的速度。

在SCCNet的實作上，因為考慮到最後的Fully Connect層需要先去計算可能輸入的大小，因此在SSCNet上還有寫計算Conv的輸出和pooling層的輸出和最後要給Fully Connect，以下為三者的計算方式。

Conv Layer：

：Conv Layer的輸入高和寬與輸出高和寬

：Conv Layer的feature map大小 ()

：Conv Layer 的stride 大小 ()

：Conv Layer 的 padding 大小 ()

Pooling Layer：

：Pooling Layer的輸入高和寬與輸出高和寬

：Pooling Layer的feature map大小 ()

：Pooling Layer的stride 大小 ()

：Pooling Layer的 padding 大小 ()

FC Layer：

：對應到Flatten之後要算進入FC的大小

**Anything you want to mention**

在SCCNet的實作階段，有發現於論文中有提到，第一層Conv做完之後需要做permute的動作，但是在程式碼的實作上直接將第一層輸出對應到第二層的輸入，而在對第二層Conv的kernel size做轉換也能達到同樣訓練的效果，並且針對第二層的feature map，去計算的話兩者的參數量是一致的。

Analyze on the experiment results

**Discover during the training process**

在訓練時SD時，發現到不同learning rate、Nu、Nt、optimizer、batch\_size取用下的差異。

* 不同的learning rate：

一張含有 文字, 螢幕擷取畫面, 鮮豔, 行 的圖片

自動產生的描述

圖.1：Nu=22、Nt=1下不同learning\_rate的最高準確度

一張含有 圖表, 文字, 行, 繪圖 的圖片

自動產生的描述

圖.2：Nu=22、Nt=1下optimizer為adamw的不同learning\_rate的準確度

一張含有 文字, 圖表, 繪圖, 行 的圖片

自動產生的描述

圖.3：Nu=22、Nt=1下optimizer為SGD的不同learning\_rate的準確度

Learning rate越小的model訓練起來準確度越高、越穩定，而adamw對於learning rate很敏感，如果learning rate太高從圖.2中可以看到0.001與0.005的波動是比圖.3的SGD還要來的大，且不穩定的。此外從圖.1來看，參數調整為Nu=22、Nt=1時不同learning rate的準確度都位於60%上下。

* 不同的 Optimizer

一張含有 文字, 螢幕擷取畫面, 鮮豔, Rectangle 的圖片

自動產生的描述

圖.4：Nu=22、Nt=1下不同optimizer的最高準確度

一張含有 文字, 圖表, 繪圖, 行 的圖片

自動產生的描述

圖.5：Nu=22、Nt=1下訓練時不同optimizer的準確度

在圖.5中可以看到adamw的optimizer一開始的訓練效果就能達到 60上下，但是因為learning rate太大的關係，adamw沒法穩定的保持在準確度60以上。由圖.4也可以看出在同樣的參數下訓練出來的結果都差不多

* 不同的batch大小

一張含有 文字, 螢幕擷取畫面, 鮮豔, 繪圖 的圖片

自動產生的描述

圖.6：Nu=22、Nt=1時不同batch\_size的最高準確度

一張含有 文字, 圖表, 繪圖, 行 的圖片

自動產生的描述

圖.7：Nu=22、Nt=1下訓練時不同batch\_size的準確度

由上圖.6和圖.7中可以看到，adamw在batch越大時表現的效果越好，相反的sgd在batch大小越小時，準確率越高。

* 不同的Nu

一張含有 文字, 螢幕擷取畫面, 鮮豔, 行 的圖片

自動產生的描述

圖.8：batch\_size=8時不同參數的準確度

一張含有 文字, 圖表, 行, 繪圖 的圖片

自動產生的描述

圖.9：batch\_size=8、Nt=1、optimizer=adamw、learning\_rate=0.001

在訓練時不同Nu的準確度

一張含有 文字, 圖表, 繪圖, 行 的圖片

自動產生的描述

圖.10：batch\_size=8、Nt=1、optimizer=SGD、learning\_rate=0.001

在訓練時不同Nu的準確度

在圖.9和圖.10中可以看到，Nu在一定的數量之後訓練到的結果都落在準確度60上下，但唯獨Nu為2時的效果不佳(如圖.8所示)，此外在圖.8和圖.10也有發現在使用SGD作為optimizer的情況下Nu為44、Nt為1時訓練結果是最好的。

* 不同的Nt

一張含有 文字, 螢幕擷取畫面, 鮮豔, 平行 的圖片

自動產生的描述

圖.11：batch\_size=8時不同參數的準確度

一張含有 文字, 圖表, 行, 繪圖 的圖片

自動產生的描述

圖.12：batch\_size=8、Nu=22、optimizer=adamw、learning\_rate=0.001

在訓練時不同Nt的準確度

在上圖.12中，Nt為1或2時的準確度非常相近，但是隨著Nt數量的增加準確度也跟著一起下降。

對於SD訓練的總結，在optimizer的選擇上使用SGD時batch大小越低越好，反之選擇adamw的batch大小則是越高越好，在learning rate的比較下learning rate對adamw 的影響是比較大的。Nu和Nt的比較下，Nu超過11之後的準確度都是差不多的，而Nt則是1和2測試出來的準確度都差不多，但是隨著後續數字增加，訓練出來的效果就會越來越差。

**Comparison between the three training methods**

以下是根據20多萬組epoch的數據總結下來的三個訓練結果，其中SD有7.2萬組、LOSO有2.5萬組、LOSO+FT有10萬組，透過不斷調整上述有提到的參數得到的以下結果。

一張含有 文字, 螢幕擷取畫面, Rectangle, 圖表 的圖片

自動產生的描述

其中SD拿到最高的準確度為65.23%、LOSO為63.54%、LOSO+FT為80.90%。

SD使用的參數為Nu=44、Nt=1、batch\_size=8、learning\_rate=0.001、optimizer=SGD。

LOSO使用的參數為Nu=22、Nt=1、batch\_size=8、learning\_rate=0.001、optimizer=AdamW

LOSO+FT使用的參數為Nu=22、Nt=1、batch\_size=32、learning\_rate=0.001、optimizer=adamw挑選的LOSO原先訓練集為Nu=22、Nt=1、batch\_size=16、learning\_rate=0.005、optimizer=SGD，在訓練一定準確度的基礎上轉移訓練，改成使用大的batch size和adamw 的optmizier。

**Anything you want to mention**

在做LOSO-FT訓練時，挑選在training時loss較高的會比較容易train起來，但是如果太高或者準確率太低，會導致最終準確率突破不了80%，換句話說，挑選(針對測試集)準確度高的但是(訓練集)尚未收斂(loss高)的記錄檔進行finetune的轉移訓練，效果會比已經收斂的記錄檔還要來的好。

Discussion

**What is the reason to make the task hard to achieve high accuracy**

根據EEG資料集的論文，輸入的資料有兩個session，s1T和s1E，且這兩個session都在不同天去蒐集資料，以subject 1為例(s1T/s1E)，T為training資料，E則是Evaluation資料。每一個session中包含了 6 個偵測回合，每一回合(run)都需要分別偵測4種不同的動作(classes)：左手、右手、雙腳、舌頭，而每一個種類的動作都需要偵測12次試驗(trail)， 6次48個試驗，總共是288個試驗。為了採集這些動作的特徵點，使用了22個電極貼片，每個貼片隨機採樣438個時間點的特徵值。

而每個受試者(subject)可能都有所不同，因此導致很難一次訓練出一個每個受試者都能準確偵測的模型，外在干擾的因素過多(包含兩個session於不同天的偵測問題)。

**What can you do to improve the accuracy of this task?**

透過finetune的方式先從訓練好的資料再次訓練，但是使用其他訓練集，再次訓練會達到更好得效果。

在訓練中，透過不同的調整參數，發現透過調整Nu的大小可以提升model對不同subject之間的辨識度，訓練得到的結果Nu=22比Nu=2時還要來的好。同樣的Nt在數值越小的情況訓練的效果越好。

也可以透過選擇不同的optimizer讓整體的準確度更高，也可以調整learning rate讓他訓練時嘗試收斂在更好的地方。